SAE15 :

TRAITEMENT DE DONNEES

HEMMI Anass
UNIVERSITE DE HAUTE-ALSACE 34 rue du GRILLENBREIT, 68000 COLMAR

TACHE A EFFECTUER

```python
import numpy as np

# Initialisation de la graine aléatoire
np.random.seed(0)

# Tirage aléatoire des 5 numéros gagnants
winning_numbers = np.random.randint(1, 46, size=5)

print("Les numéros gagnants sont:", winning_numbers)
```

1- Dans les tirages de loterie, la prédiction est la valeur moyenne attendue des numéros tirés. Cela peut être calculé en divisant la somme de tous les nombres possibles (1 à 45) par le nombre total de nombres possibles (45).

2- Des formules combinatoires peuvent être utilisées pour déterminer la probabilité de tirer une combinaison particulière. Le premier nombre a des options de 45, le deuxième 44, le troisième 43, le quatrième 42 et le cinquième 41. Au total il y a 45x44x43x42x41 combinaisons possibles. Cela vous donne 45 x 44 x 43 x 42 x 1 chance sur 41 de toucher une combinaison particulière.

3- Pour s'assurer que le programme simule correctement les tirages de loterie, les valeurs attendues obtenues par le programme peuvent être comparées aux valeurs attendues théoriques. Pour ce faire, nous devons collecter les résultats de plusieurs tirages et calculer la moyenne. Si cette moyenne est proche de la valeur théorique attendue, cela indique que le programme simule correctement le tirage de la loterie. Il est important de noter qu'un grand nombre de tirages doivent être effectués pour obtenir une précision suffisante pour la comparaison.

PARTIE PROBABILITE

EXERCICE 3 :

Pour un tiercé avec 15 chevaux engagés dans la course, il y a 15 parmi 3 soit (15!)/(3!*(15-3)!) =455 combinaisons possibles. Cela signifie qu'il y a 455 combinaisons différentes de 3 chevaux qui peuvent être choisies pour un pari de tiercé.
Si un parieur prend 5000 paris différents pour le tiercé de l'après-midi, il ne peut pas être déduit combien de chevaux engagés dans la course. Il est possible que le parieur ait pris des paris sur plusieurs courses ou il a pris des paris multiples sur la même course.

EXERCICE4 :

(a) La probabilité d'avoir uniquement des faces en lançant cinq pièces de monnaie est de (1/2)^5 = 1/32.

(b) La probabilité d'avoir exactement trois faces en lançant cinq pièces de monnaie est de 5C3 * (1/2)^3 * (1/2)^2 = 10 * (1/32) = 10/32

(c) La probabilité d'avoir au moins trois faces en lançant cinq pièces de monnaie est :

P(au moins 3 faces) = 1 - P(au plus 2 faces) = 1 - ((5!/(0!5!))(1/2)^0*(1/2)^5 + (5!/(1!4!))(1/2)^1*(1/2)^4 + (5!/(2!3!))(1/2)^2*(1/2)^3) = 1- (1/32 + 5/32 + 10/32) = 1-16/32 = 16/32 = 0.5

PROBABILITE RELATIVE AU LOTO.

Pour ce jeu du loto, où l'ordre est pris en compte, il y a 45 options pour chaque numéro, donc le nombre de combinaisons possibles est 45! / (45-5)! = 146611080

Pour ce jeu du loto, où l'ordre n'est pas pris en compte, il y a 45 options pour chaque numéro. Le nombre de combinaisons possibles est donc 45C5 (45 combinaisons possibles pour 5 boules) = 1221759.0

Pour avoir les 5 bon numéros (quel que soit l'ordre), la probabilité est de 1/ 1221759.0

Pour avoir les 5 bon numéros en respectant l'ordre du tirage, la probabilité est de 1/146611080

PARTIE ALGORITHME ET PROGRAMMATION

ALGORITHMES ET PROGRAMMES RELATIFS AU TRI :

PROGRAMME TRI PAR INSERTION :

L'algorithme de tri par insertion est un algorithme de tri itératif qui trie les éléments d'un tableau en insérant chaque élément dans sa position appropriée parmi les éléments triés. Elle est constituée de deux phases: Il itère sur les éléments de la liste, en partant du deuxième élément (index 1) jusqu'au dernier élément. Pour chaque élément, il compare l'élément courant avec les éléments précédemment triés et insère le plus grand élément à sa place en le décalant vers la droite.

```
def cocktail_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        swapped = False
        for j in range(i, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:
            break
```

```
        swapped = False
        for j in range(n - 2 - i, i, -1):
            if arr[j] < arr[j - 1]:
                arr[j], arr[j - 1] = arr[j - 1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr
```

PROGRAMME TRI PAR INSERTION :

L'algorithme de tri par insertion est un algorithme de tri itératif qui trie les éléments d'un tableau en insérant chaque élément dans sa position appropriée parmi les éléments triés. Elle est constituée de deux phases: Il itère sur les éléments de la liste, en partant du deuxième élément (index 1) jusqu'au dernier élément. Pour chaque élément, il compare l'élément courant avec les éléments précédemment triés et insère le plus grand élément à sa place en le décalant vers la droite.

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key_item = arr[i]
        j = i-1
        while j >=0 and key_item < arr[j] :
                arr[j + 1] = arr[j]
                j -= 1
        arr[j + 1] = key_item
    return arr
```

PROGRAMME TRI PAR FUSION :

Le tri par fusion est un algorithme de tri qui utilise une méthode de division pour régner. Cela implique de diviser de manière récursive un tableau en sous-tableaux jusqu'à ce qu'ils ne contiennent qu'un seul élément, puis de fusionner les sous-tableaux triés pour obtenir le tableau trié final. Il se compose de deux étapes principales : Diviser un tableau en deux sous-tableaux de taille égale Fusionnez deux sous-tableaux triés pour obtenir le tableau trié final.

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = merge_sort(arr[:mid])
        R = merge_sort(arr[mid:])
        return merge(L, R)
    return arr


def merge(L, R):
    res = []
    i = j = 0
```

```
    while i < len(L) and j < len(R):
        if L[i] < R[j]:
            res.append(L[i])
            i += 1
        else:
            res.append(R[j])
            j += 1
    res += L[i:]
    res += R[j:]
    return res
```

RECHERCHE DICHOTOMIQUE :

SAUVEGARDE ET CHARGEMENT DES DONNEES :

FORMAT BINAIRE :

Un format binaire est un format de données qui stocke des informations à l'aide de codes binaires (0 et 1). Il est souvent utilisé pour stocker des données pour le traitement informatique car il est plus efficace de stocker et de transférer. Cependant, les données stockées sous forme binaire sont souvent difficiles à lire et à comprendre pour les humains. Un format lisible par l'homme est un format de données qui stocke des informations à l'aide de caractères lisibles par l'homme, tels que des lettres, des chiffres et des symboles. Il est généralement utilisé pour stocker des données que les humains doivent lire et comprendre, telles que des documents texte et des feuilles de calcul.

```
import pickle
# Création d'un dictionnaire contenant les informations d'un utilisateur
data = {
    'nom': 'HEMMI',
    'prenom': 'Anass',
    'age': 19,
    'ville': 'Mulhouse',
}


def sauvegarder_donnees_binaires(donnees, nom_fichier):
    with open(nom_fichier, 'wb') as fichier:
        pickle.dump(donnees, fichier)


# Sauvegarde des données dans un fichier
sauvegarder_donnees_binaires(data, 'données.binaires')


def charger_donnees_binaires(nom_fichier):
```

```
    with open(nom_fichier, 'rb') as fichier:
        donnees = pickle.load(fichier)
    return donnees


# Chargement des données d'un utilisateur à partir d'un fichier
donnees_utilisateur = charger_donnees_binaires('données.binaires')

# Affichage des données de l'utilisateur
print(donnees_utilisateur)
```

FORMAT JSON :

```
import json
import os

nom_du_fichier = "C:\\Users\\33656\\Desktop\\SAE.json"


def sauvegarder_donnees_json(donnees, nom_fichier):
    with open(nom_fichier, 'w') as fichier:
        json.dump(donnees, fichier)
    print("Fichier enregistré dans le répertoire : ", nom_du_fichier)


# Création d'un dictionnaire contenant les informations d'un utilisateur
utilisateur = {
    'nom': 'HEMMI',
    'prenom': 'Anass',
    'age': 19,
    'ville': 'Paris',
}

# Sauvegarde des données dans un fichier
sauvegarder_donnees_json(utilisateur, nom_du_fichier)
```

FORMAT CSV :

```
import csv
import os


def sauvegarder_donnees_csv(donnees, nom_fichier, nom_colonnes):
```

```
    # Création du répertoire s'il n'existe pas
    if not os.path.exists(os.path.dirname(nom_fichier)):
        os.makedirs(os.path.dirname(nom_fichier))
    # Écriture des données dans le fichier CSV
    with open(nom_fichier, 'w', newline='') as f:
        writer = csv.writer(f)
        writer.writerow(nom_colonnes)
        for data in donnees:
            writer.writerow(data)
    print("Fichier enregistré dans le répertoire : ",
          os.path.dirname(nom_fichier))


# Création d'une liste contenant les informations d'utilisateur
utilisateurs = [{'nom': 'HEMMI', 'prénom': 'Anass',
                 'age': 19, 'ville': 'Mulhouse'}]

# Sauvegarde des données dans un fichier
nom_fichier = "C:\\Users\\33656\\Desktop\\SAE.csv"
nom_colonnes = ['nom', 'prénom', 'age', 'ville']
sauvegarder_donnees_csv(utilisateurs, nom_fichier, nom_colonnes)
```

VISUALISATION DE LA DISTRIBUTION DES NUMEROS TIRES :

PRESENTATION DE L'HISTOGRAMME :

L'histogramme est un type de graphique utilisé pour répartir les données.

Il comprend une barre verticale représentant les occurrences d'une variable quantitative.

Les barres sont généralement placées les unes à côté des autres, les x représentant les types de données et l'axe des y représentant la fréquence.

Voici un exemple d'algorithme pour calculer un histogramme des numéros sortis :

```
procédure calculer_histogramme(nums_sortis: tableau[0..n] de entier)
    // Initialiser un tableau de fréquences à zéro
    déclarer freq[0..max(nums_sortis)] de entier
    pour i allant de 0 à max(nums_sortis) faire
        freq[i] <- 0
    fin pour
    // Compter les fréquences de chaque numéro
    pour i allant de 0 à n-1 faire
        freq[nums_sortis[i]] <- freq[nums_sortis[i]] + 1
    fin pour
```

```
    // Afficher l'histogramme
    pour i allant de 0 à max(nums_sortis) faire
        afficher i et freq[i] sous forme de barre verticale
    fin pour
fin
```

IMPLEMENTATION PYTHON :

```python
from collections import defaultdict
import matplotlib.pyplot as plt

nums_sortis = [2, 5, 6, 8, 2, 8, 2, 1, 7, 5, 6, 3, 2, 6, 8, 9, 2, 5, 2, 4]


def calculer_histogramme(nums_sortis):
    # Initialiser un tableau de fréquences à zéro
    freq = defaultdict(int)
    for num in nums_sortis:
        freq[num] += 1
    # Afficher l'histogramme
    plt.bar(freq.keys(), freq.values())
    plt.show()


calculer_histogramme(nums_sortis)
```

TRACAGE HISTOGRAMME PYTHON (MATPLOTLIB):

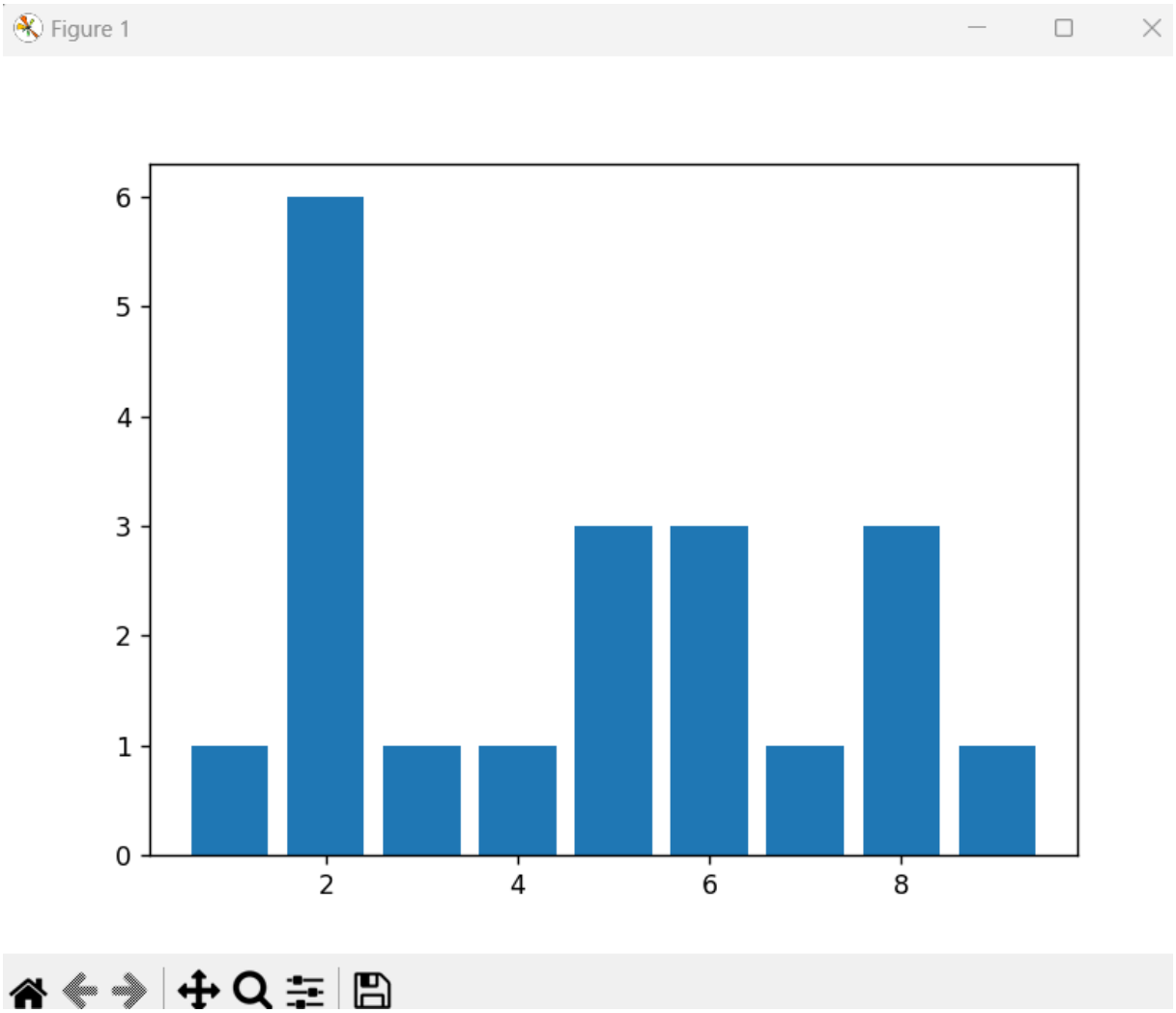
```python
import matplotlib.pyplot as plt
from collections import defaultdict


nums_sortis = [2, 5, 6, 8, 2, 8, 2, 1, 7, 5, 6, 3, 2, 6, 8, 9, 2, 5, 2, 4]


def tracer_histogramme(nums_sortis):
    # Compter les fréquences de chaque numéro
    freq = defaultdict(int)
    for num in nums_sortis:
        freq[num] += 1
    # Afficher l'histogramme en utilisant bar()
    plt.bar(freq.keys(), freq.values())
```

```
    plt.show()


tracer_histogramme(nums_sortis)
```



Vous pouvez voir que l'histogramme des numéros tirés devient plus plat lorsque le nombre de tirages est élevé. Cela signifie que la fréquence de chaque chiffre tiré sera approximativement égale. Autrement dit, chaque numéro a une probabilité égale d'être tiré.